Branch-and-Bound e Aproximações: Desafios na Computação de Rotas para o Caixeiro Viajante

Lucas Almeida Santos de Souza[1]

[1]Departamento de Ciência da Computação
Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brasil

lucasalmeida@dcc.ufmg.br

Abstract. *This paper assesses the performance of three implementations of algorithms for computing routes in the NP-hard Traveling Salesman Problem. Two approximate solutions and one exact solution will be analyzed, evaluating the time and space used by each, as well as the quality of the solution.*

Resumo. *Este artigo avalia o desempenho de três implementações de algoritmos para computação de rotas no problema NP-difícil do Caixeiro Viajante. Serão analisadas duas soluções aproximadas e uma exata, avaliando o tempo e espaço usado por cada uma, além da qualidade da solução.*

1. Apresentação do Problema

O Problema do Caixeiro Viajante (PCV) ou Traveling Salesperson Problem (TSP) é um desafio NP-difícil clássico de otimização combinatória, importante na ciência da computação teórica e na pesquisa operacional. Formalizado pela primeira vez por W.R. Hamilton em 1835, o problema apresenta um viajante que deve visitar um conjunto de cidades exatamente uma vez, retornando à cidade de origem, com o objetivo de minimizar a distância total percorrida.

Este artigo se propõe a explorar e comparar o desempenho de três abordagens notáveis para resolver o Problema do Caixeiro Viajante: o método ”Twice Around the Tree”, a heurística de ”Christofides” e uma solução baseada em ”Branch and Bound”. Cada uma dessas técnicas aborda o desafio de forma única, apresentando vantagens e limitações que serão discutidas nas próximas seções.

2. Implementação

Nessa seção será discutida a implementação dos algoritmos para solução do PCV. Ambos os algoritmos aproximativos exigem que o grafo de entrada respeite a métrica de *desigualdade triangular*, ou seja, para quaisquer arestas $(u, v), (v, w), (w, u)$, temos que

$$c((u, v)) + c((v, w)) \geq c((u, w)).$$

Além disso, o grafo deve ser completamente conexo. Uma entrada para o Problema do Caixeiro Viajante nesse formato é chamada Metric-TSP.

O algoritmo Branch and Bound, por outro lado, permite qualquer tipo de grafo, pois se trata de uma solução genérica para o problema.

2.1. Especificações

Os algoritmos aproximativos foram implementados na linguagem Python 3.10.12, utilizando a biblioteca Networkx[1] para armazenar as entradas em uma estrutura de dados grafo. O algoritmo Branch and Bound foi implementado na linguagem C++17, utilizando bibliotecas de estruturas de dados STL para armazenar o grafo como uma matriz de adjacência ponderada.

2.2. Twice around the tree

O algoritmo "Twice around the tree" (TATT), também conhecido como Approx-TSP-Tour [Cormen et al. 2009], é uma abordagem 2-aproximada de complexidade $O(|V|^2)$. O algoritmo tem três passos:

1. Primeiramente, construímos uma Árvore Geradora Mínima (AGM).
2. Depois, duplicamos todas as arestas da AGM e geramos um circuito euleriano[2].
3. Por fim, removemos vértices repetidos do caminho, transformando arestas u-v-w em u-w para todo vértice repetido v, e adicionamos o primeiro vértice ao final do caminho para transformá-lo em um ciclo.

Os passos 2 e 3 podem ser feitos também através de um caminho DFS em pré-ordem pela AGM, anotando os vértices assim que são visitados.

2.3. Christofides

O algoritmo de Christofides [Goodrich and Tamassia 2014] é uma solução que acrescenta alguns passos à abordagem Twice around the tree para tornar o algoritmo 1.5-aproximado, apesar de tornar sua complexidade $O(|V|^3)$. Ele foi proposto inicialmente pelos matemáticos Nicos Christofides e Anatoliy I. Serdyukov em 1976. O algoritmo adiciona dois passos a mais em relação ao anterior:

1. Iniciamos construíndo a Árvore Geradora Mínima (AGM).
2. Depois selecionamos os vértices da árvore que têm grau ímpar, que impossibilitariam o caminho euleriano.
3. A partir do subgrafo induzido do grafo original com os vértices selecionados no passo anterior, buscamos um Perfect Matching de custo mínimo. Isso faz com que as arestas que serão adicionadas à AGM sejam as menores possíveis.
4. Após adicionar as arestas do Perfect Matching à AGM, podemos prosseguir como no algoritmo anterior, gerando um circuito euleriano.
5. Por fim, removemos vértices repetidos do caminho, transformando arestas u-v-w em u-w para todo vértice repetido v, e adicionamos o primeiro vértice ao final do caminho para transformá-lo em um ciclo.

2.4. Branch and Bound

A solução Branch and Bound para o PCV [Levitin 2011] computa o *lower bound* da seguinte forma: Para cada cidade i, $1 \leq i \leq n$, encontre a soma s_i das distâncias das duas cidades mais próximas de i, ou seja, as menores arestas incidentes a i. Some esse

[1] https://networkx.org/, acesso em 2023-12-09

[2] Circuito Euleriano é um caminho que passa por todas as arestas de um grafo exatamente uma vez, podendo repetir vértices. É necessário duplicar as arestas da AGM para permitir que o circuito seja feito.

valor para todas as cidades e divida o resultado por 2, pois queremos uma aproximação de uma solução válida, onde existem apenas n arestas. Caso o resultado seja fracionário, arredonde para cima.

$$lb = \left\lceil \frac{s}{2} \right\rceil$$

Ao escolher uma aresta para entrar na solução, o lower bound é modificado da seguinte forma: Se a aresta (a, d) foi escolhida para uma solução parcial, o lower bound é computado somando as duas menores arestas incidentes em cada vértice, porém substituindo uma das arestas de a e uma das arestas de d pelas arestas (a, d) e (d, a), respectivamente.

O algoritmo de Branch and Bound percorre a ***state space tree*** utilizando a estratégia ***best-first branch-and-bound***. Ou seja, em cada nível da árvore, ele analisa o lower bound de cada filho para explorar o mais promissor, utilizando uma fila de prioridade ordenada pelo menor lower bound. Essa forma de se realizar o algoritmo é bem mais eficiente do que explorar em busca profunda, porém é mais suscetível a problemas de memória para casos maiores, onde armazenar o estado de todos os filhos em uma estrutura de dados se torna inviável.

3. Resultados

Nesta seção, serão analisados os resultados dos testes dos algoritmos, seguindo três métricas: tempo de execução, espaço ocupado e qualidade da solução em relação à solução ótima. Para os experimentos, foram utilizados ao total 78 instâncias de testes da biblioteca TSPLIB[3], uma biblioteca com diversas instâncias para o Problema do Caixeiro Viajante, cada uma com uma quantidade diferente de vértices. Os últimos casos de teste não puderam ser executados devido a problemas com a função de medição de uso da memória, que não soube lidar com as entradas maiores.

Dado que o algoritmo Branch and Bound não conseguiu calcular o menor caso de teste dentro do prazo estabelecido de 30 minutos, apesar de ter sido implementado em C++ para otimização, esta seção será focada exclusivamente no desempenho dos dois algoritmos aproximativos. Estes, mesmo implementados em Python, conseguiram executar os testes dentro do tempo determinado.

3.1. Análise de tempo

Para medir o tempo dos algoritmos, foi usada a biblioteca `time` do python. Foi guardado um timestamp logo antes e um logo após a chamada de cada função, para que o pré processamento dos dados não afetasse a medida de tempo de execução. Os resultados podem ser vistos na Figura 1.

Podemos ver que, apesar do fator de aproximação maior, o algoritmo TATT tem um tempo de execução consideravelmente menor. Isso evidencia o impacto da adição do Perfect Matching no algoritmo de Christofides. Apesar de haverem alguns *outliers*, os algoritmos aproximativos parecem seguir um padrão bem claro na relação do tempo com o tamanho da entrada.

[3]http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/, acesso em 2023-12-09

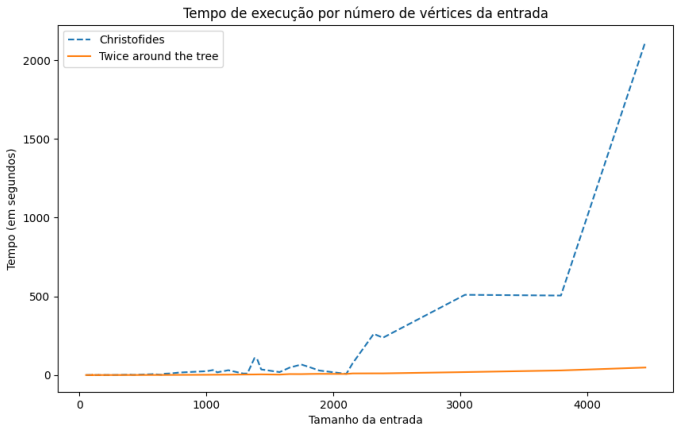

Figure 1. O gráfico mostra a relação entre o número de vértices do grafo de entrada e o tempo de execução de cada um dos algoritmos.

3.2. Análise de espaço

Para analisar a memória utilizada, foram utilizadas a biblioteca `memory_profiler` do python. Devido ao impacto que a função `memory_usage` pode ter no tempo de execução, a função foi chamada novamente na análise de espaço. Os parâmetros da função foram configurados para receber o maior uso de memória registrado durante a execução da função. Os resultados são vistos na Figura 2.

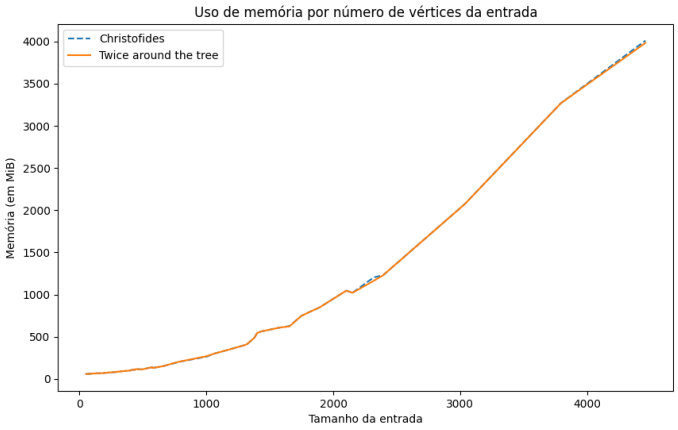

Figure 2. O gráfico mostra a relação entre o número de vértices do grafo de entrada e a memória utilizada por cada um dos algoritmos.

Pelo gráfico, podemos afirmar que não há nenhuma diferença significativa entre os algoritmos aproximativos em relação ao uso de memória. Isso é evidente, pois a única diferença é a realização do perfect matching dos vértices de grau ímpar, que pode ser feito sem precisar de uma cópia do grafo.

3.3. Análise de qualidade

Para medir a qualidade das soluções, apenas os algoritmos aproximativos foram levados em consideração, pois o algoritmo Branch and Bound sempre resulta na solução ótima. O cálculo é feito simplesmente dividindo a solução do algoritmo para uma instância

pela solução ótima dessa mesma instância, para chegar no fator de aproximação daquela execução. Os resultados são vistos na Figura 3

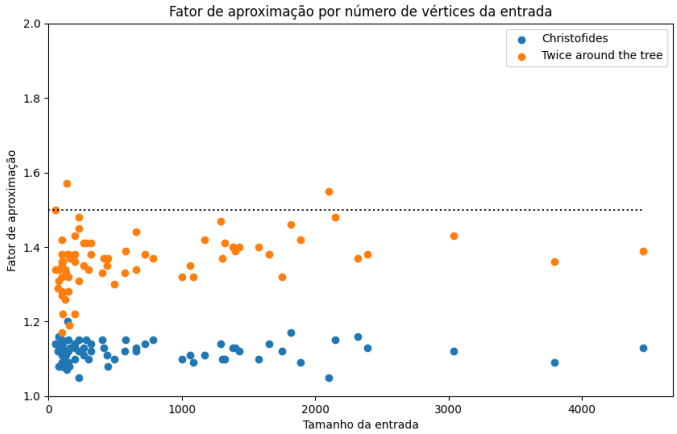

Figure 3. O gráfico mostra a relação entre o número de vértices do grafo de entrada e o fator de aproximação da solução.

Esses resultados se mostram um pouco inconsistentes em relação ao tamanho da entrada, mas é evidente que o algoritmo de Christofides tem um fator de aproximação menor que o TATT para todos os casos. Além disso, podemos ver que os algoritmos mantiveram seus limites teóricos de aproximação de 2-aproximado e 1.5-aproximado para TATT e Christofides, respectivamente, em todos os casos de teste.

4. Conclusões

Este artigo realiza uma análise comparativa entre algoritmos para resolver o Problema do Caixeiro Viajante (PCV), abrangendo dois algoritmos aproximativos e um algoritmo exato. Os algoritmos aproximativos analisados são o Twice Around the Tree (TATT), também conhecido como Approx-TSP-Tour, e o algoritmo de Christofides, com fatores de aproximação de 2 e 1.5, respectivamente. O artigo destaca que, embora o Christofides ofereça soluções mais próximas da exatidão, o TATT apresenta resultados quase tão precisos com um tempo de execução significativamente inferior, mantendo-se quase sempre abaixo do 1.5-aproximado em testes, apesar de seu limite teórico ser 2-aproximado.

O algoritmo Branch and Bound, embora não consiga lidar com instâncias maiores, demonstra eficácia ao produzir resultados exatos para casos de teste menores. Apesar de não serem apresentadas métricas, a implementação desse algoritmo é apresentada no artigo, destacando sua utilidade em casos que demandam precisão apesar do desafio de escalabilidade. A análise abrangente deste estudo facilita a escolha de algoritmos para resolver o PCV, levando em conta as trocas necessárias entre precisão e eficiência computacional.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). The traveling-salesman problem. In *Introduction to Algorithms*, chapter 35.2, pages 1111–1117. MIT Press, 3rd edition.

Goodrich, M. T. and Tamassia, R. (2014). The metric traveling salesperson problem. In *Algorithm Design and Applications*, chapter 18.1, pages 511–514. Wiley, 1st edition.

Levitin, A. (2011). Branch and bound. In *Introduction to The Design and Analysis of Algorithms*, chapter 12.2, pages 438–440. Addison Wesley Longman, 3rd edition.